

ぼくのかんがえる最高の レポートニング基盤

大量・多様なデータとの戦い方

AWSで実践！Analytics modernization ～事例祭り編～

みなさん、レポート作ってますか？



レポートって大変ですね。

増え続けるデータ。

拡大し続けるレポート要件。

レポートはそこら中が辛い。

場当たりに作られるクエリ。

なぜかズれるレポート。

問題をややこしくする寡等性のない集計処理。

レポートが抱える様々な課題と
どのように戦ったかを話します。

大量データには、適したアーキテクチャを。
多様なデータには、適したモデリングを。

アジェンダ

1. 自己紹介
2. レポーティング基盤について
3. 大量データとの戦い
4. 多様なデータとの戦い
5. まとめ

アジェンダ

1. 自己紹介
2. レポーティング基盤について
3. 大量データとの戦い
4. 多様なデータとの戦い
5. まとめ

自己紹介

近森 淳平 (チカモリ ジュンペイ)

@pei0804

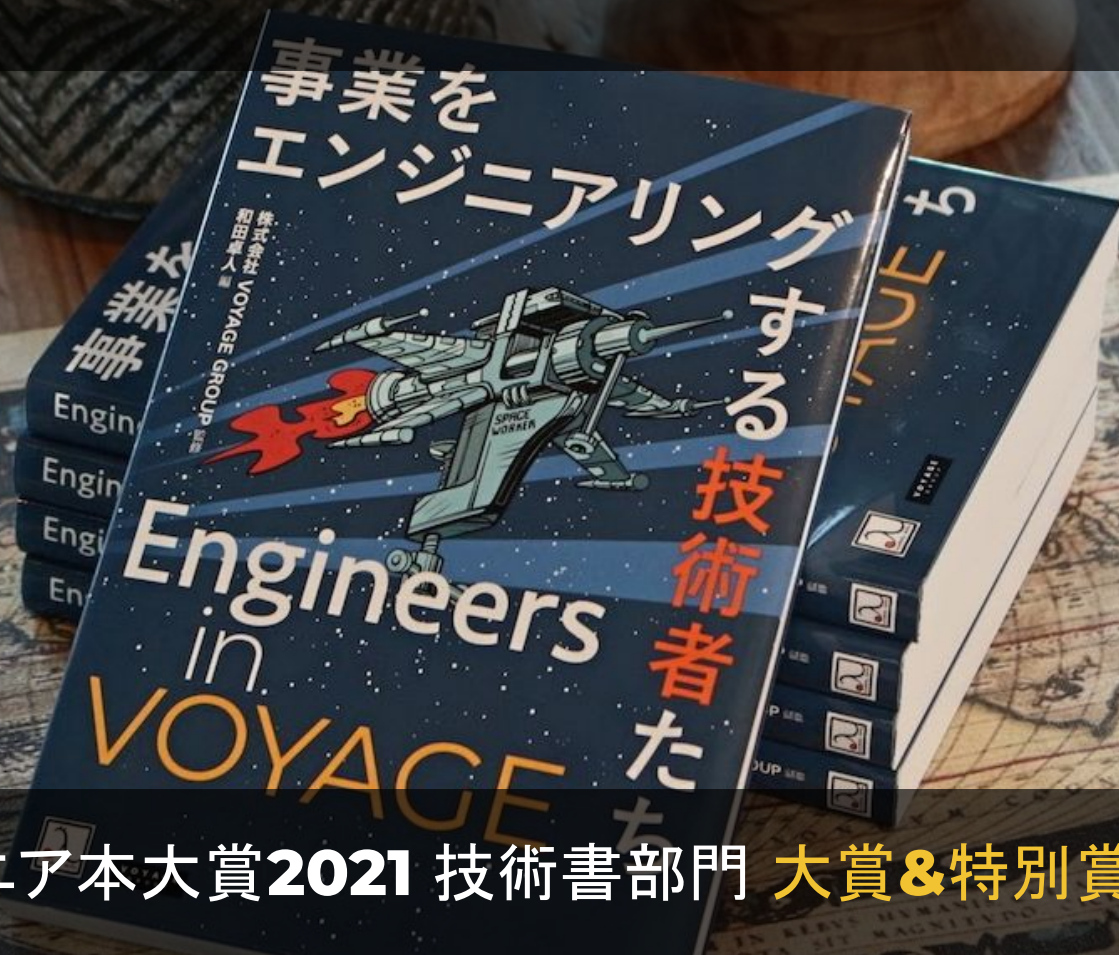


株式会社VOYAGE GROUP / Zucks エンジニア

- フルマネージドサービスに乗っかるマン。
- クライミングが好き。

What's about VOYAGE GROUP.

『Engineers in VOYAGE — 事業をエンジニアリングする技術者たち』



ITエンジニア本大賞2021 技術書部門 大賞&特別賞 受賞

VOYAGE GROUPのエンジニアとカジュアルにお話しませんか？



voyage ajiting

検索

<https://contact.voyagegroup.com/ajiting/>



アジェンダ

1. 自己紹介
2. レポーティング基盤について
3. 大量データとの戦い
4. 多様なデータとの戦い
5. まとめ

レポートティング != 分析

レポートニング != 分析

レポートニングと分析は、どちらも最終的には、組織の価値向上のために実行される。しかし、それぞれ全く異なる役割を持っている。

レポートニングは、生データを情報へ。分析は、データと情報から洞察を得るために使われる。

噛み砕くと、レポートニングは、ビジネスで何が起きているかを教えてくれる。分析は、何故それが起きているかを教えてくれる。

GitHubで開く



レポーティングと分析の違い

📅 2021.06.03に公開 ⌚ 1 min read



モデリング



データベース

データウェアハウス

スターズキーマ



Tech

ゴールは同じでも役割が違うことを理解する

レポーティングと分析が、同じ意味で使われていることを見かける。これは大きな間違いであるため、この記事で言語化することにした。（自戒の念を込めて）

レポーティングと分析は、その両方が組織の価値向上のために行われることは間違いない。しかし、期待されている役割は全く違う。結論から書くと、レポーティングは、何が起きているかを教えてくれる。分析は、何故それが起きているかを教えてくれる。

<https://zenn.dev/pei0804/articles/reporting-vs-analysis-difference>

レポーティング = What
分析 = Why, How, When

レポートティング = What
~~分析 = Why, How, When~~

レポートニング基盤について

ZucksのDSP(Demand Side Platform)の広告レポートを提供する。
例えば、以下のようなものが見れることが期待されている。

- 「昨日、何回広告表示されましたか？」
- 「今月の広告掲載費用は？」

※今日は詳しいアドテクの話をしません。

ログ -> レポート

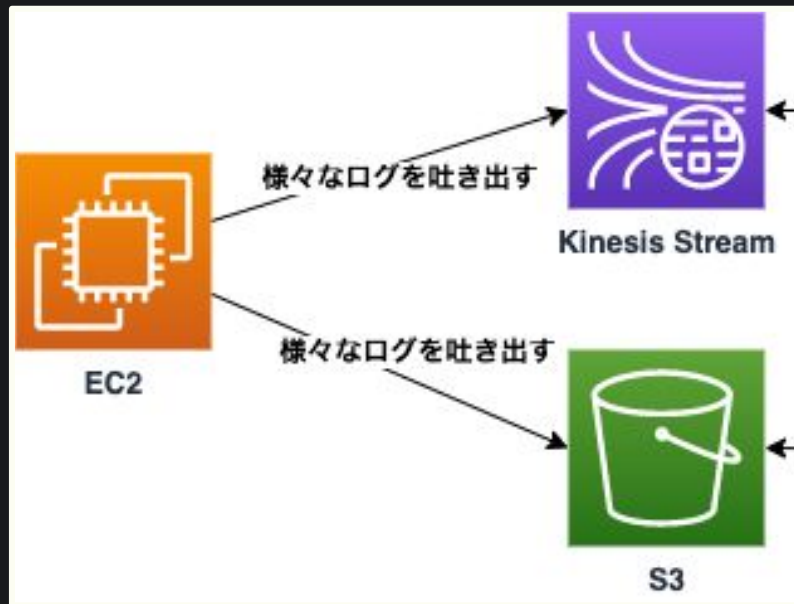
アドテクにおけるログとは

アドテクでは、**ログはお金に直結する。**

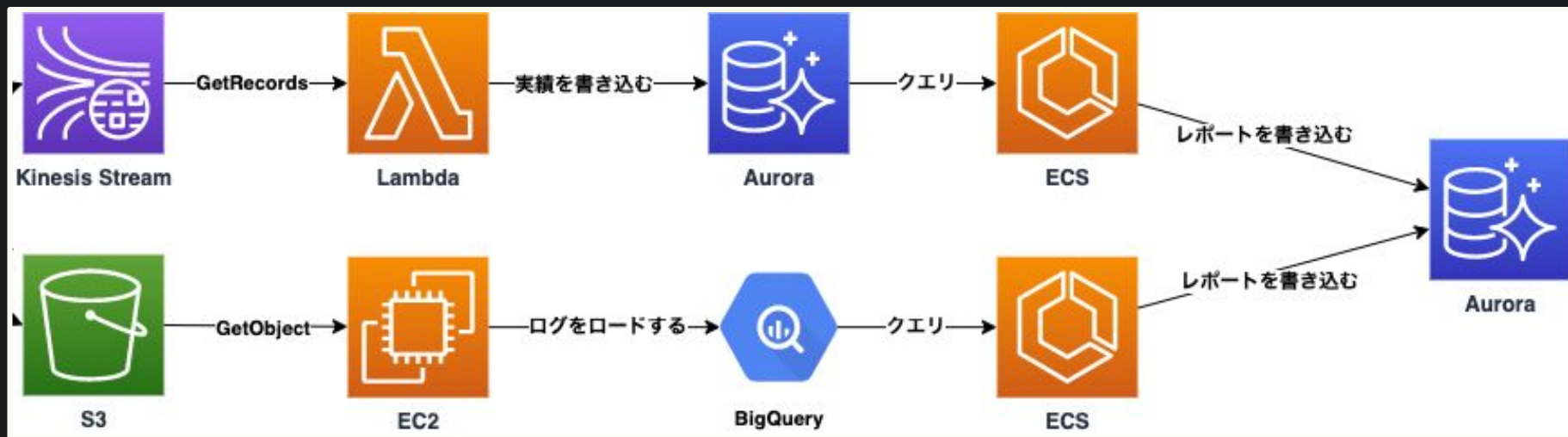
取りこぼすことは、お金を取りこぼすのと同じことになるため、
ログは丁寧に扱う必要があります。



既存のレポーティング基盤



既存のレポーティング基盤

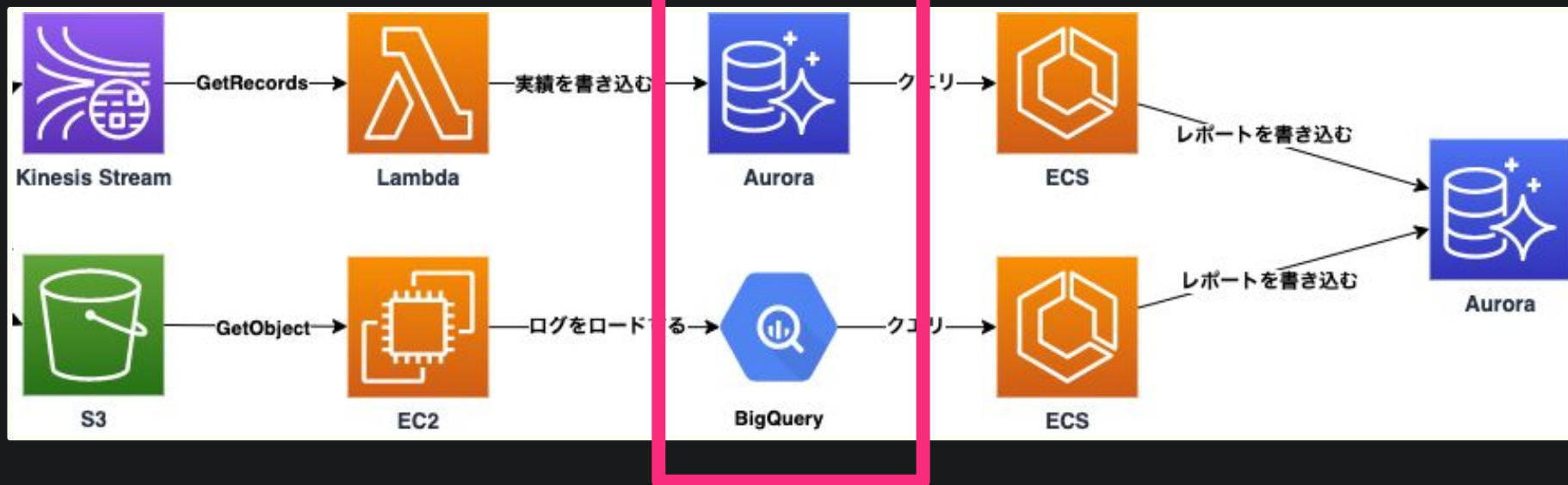


既存のレポーティング基盤の抱える課題。

アーキテクチャが辛い。

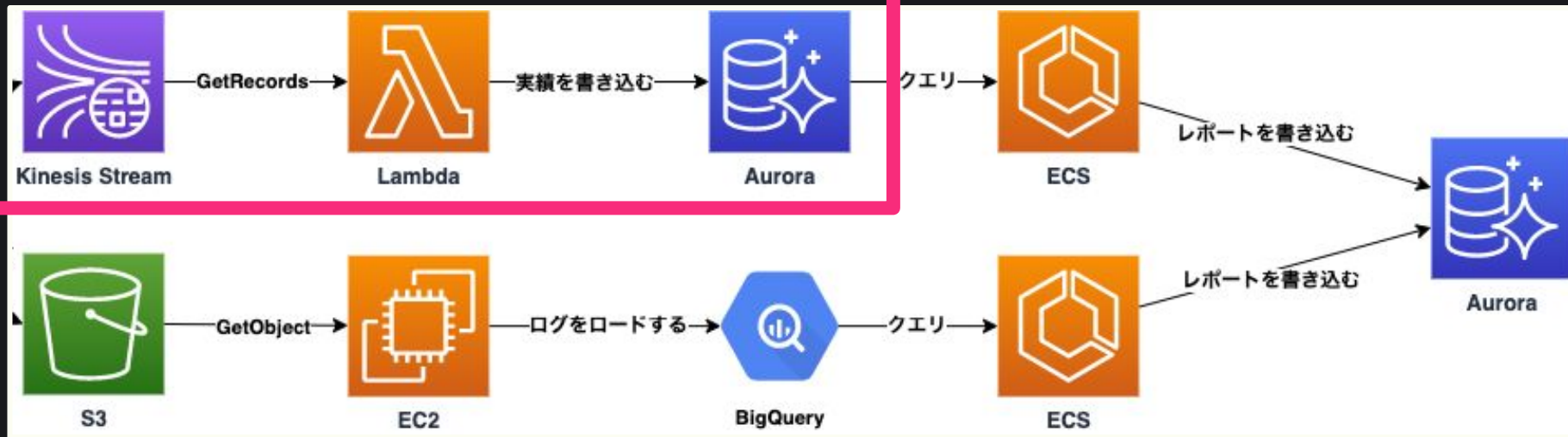
既存レポーティング基盤の課題

ソースが複数存在している。
数字が揃わないことが起きる。



既存レポーティング基盤の課題

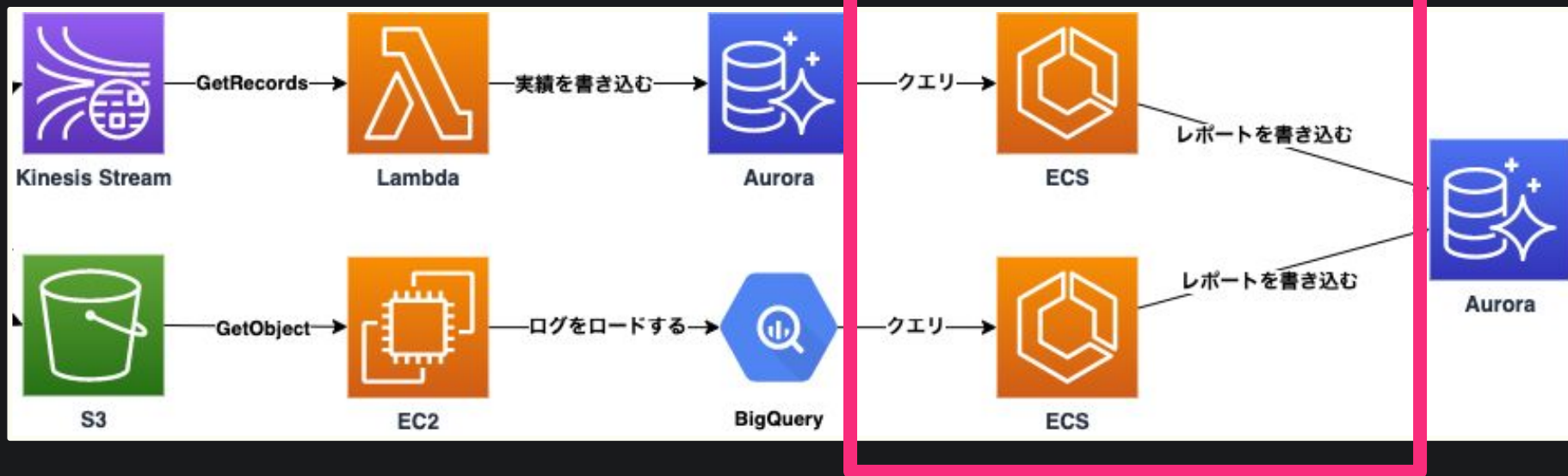
再集計が出来ない。
実績の書き込みに冪等性がない。



モデリングが辛い。

既存レポーティング基盤の課題

クエリがひたすらに辛い。



ある日、作り直すチャンスが訪れる。
これを機に、ゼロベースで作ることになる。

※ビジネス要因

開発体制

- 私

適宜、チームメンバーにレビューもらう。

アジェンダ

1. 自己紹介
2. レポーティング基盤について
3. 大量データとの戦い
4. 多様なデータとの戦い
5. まとめ

大量データとの戦い

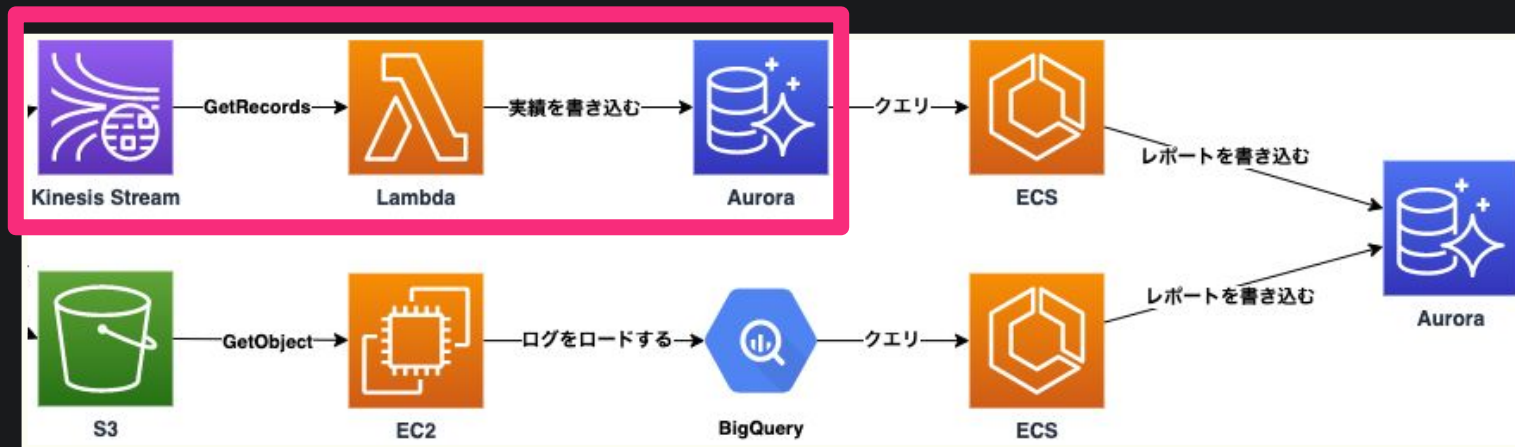
ソースとなるデータは、それなりに大量になる。

ログは種類によっては**1日で7億+-レコード**。

既存のアーキテクチャでも、ログの流量に耐えることが出来ているけど、
もっと適したアーキテクチャを考えた。

既存レポーティング基盤の見直し



実績を書き込む部分が、ラムダアーキテクチャでいう**スピードレイヤー**で組まれていた。レポーティングにおいて重要なのは正しい数字が見れることであり、**バッチレイヤー**の方が相性が良い。




ラムダアーキテクチャとは

- バッチレイヤーは、全量データに対するバッチ処理を担当する。精度の高い集計や、ロングテールをつかむ細かな集計が向く。
- スピードレイヤーは、リアルタイム処理の結果を提供する層である。直近数秒、数分、数十分のイベントの集計結果を提供することになる。

Big Data + Fast Data = ラムダアーキテクチャー !

 Tweet  いいね 1,493

 ソリューション事業部
ビッグデータ基盤ビジネスユニット 主任エンジニア
小山 哲平

はじめに

本コラムでは、ビッグデータ分析とファストデータ分析を組み合わせるための仕組みである「ラムダアーキテクチャー」の紹介をする。私どもは現在、Apache Sparkを最大限に活用したラムダアーキテクチャーの構想を練っており、その実現方式が固まった際は、コラムにてサンプルを紹介しようと考えている。ただ、「ラムダアーキテクチャー」という言葉に耳慣れない人もまだ多いかと思い、まずはラムダアーキテクチャーについての説明から始めることとする。

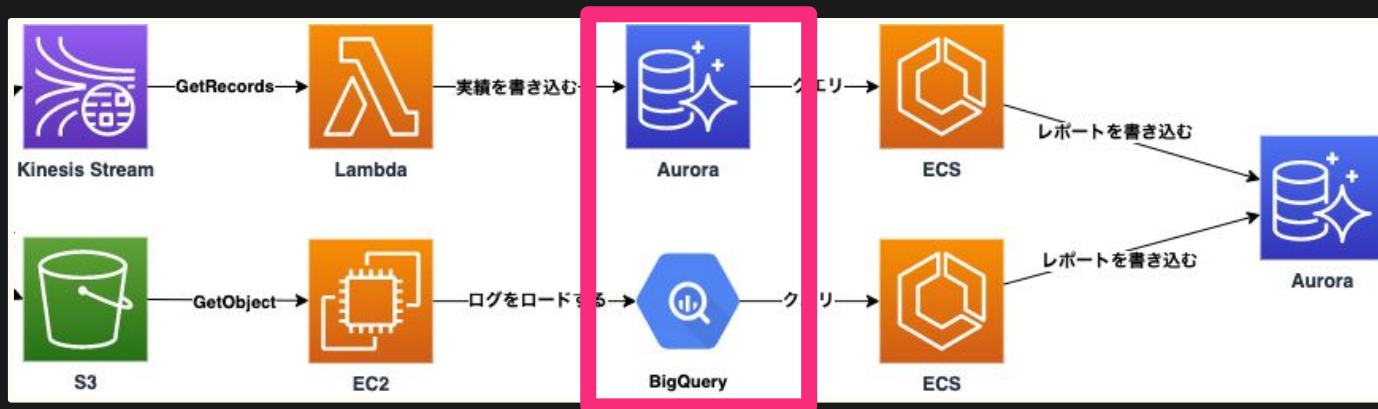
<https://www.intellilink.co.jp/column/bigdata/2015/041500.aspx>

既存レポートニング基盤の見直し

ソースを一つに集約する。

色々あって、分析用途に運用されていたBigQueryに頼っている部分があった。最初は、参考値として出していたけど、

そんな文脈は当然考慮されず、数字のズレの問い合わせがry



何故数字がズれるのか

数字がズれる主な原因は、BigQueryを使ったログ取り込み基盤が、分析向けに作られた基盤であり、確定値がほしい様なレポートに使うことを想定していないため、取り込み漏れでズれる可能性があった。例えば、クリック数でも、ソースによって違うということが起きる。

では、何故、ソースを分けるしかなかったのか？



複数ソースになった原因

実績を格納するテーブルは、X00億(定期的に削除してる)レコードくらいになっているため、カラムを追加出来なかった。

つまり、**レポート軸をあとから増やせる状態ではなくなっていた。**

これは行指向データベースの保存方法的に仕方がない。

ALTER TABLE 実績 ADD COLUMN 新しい軸 INT

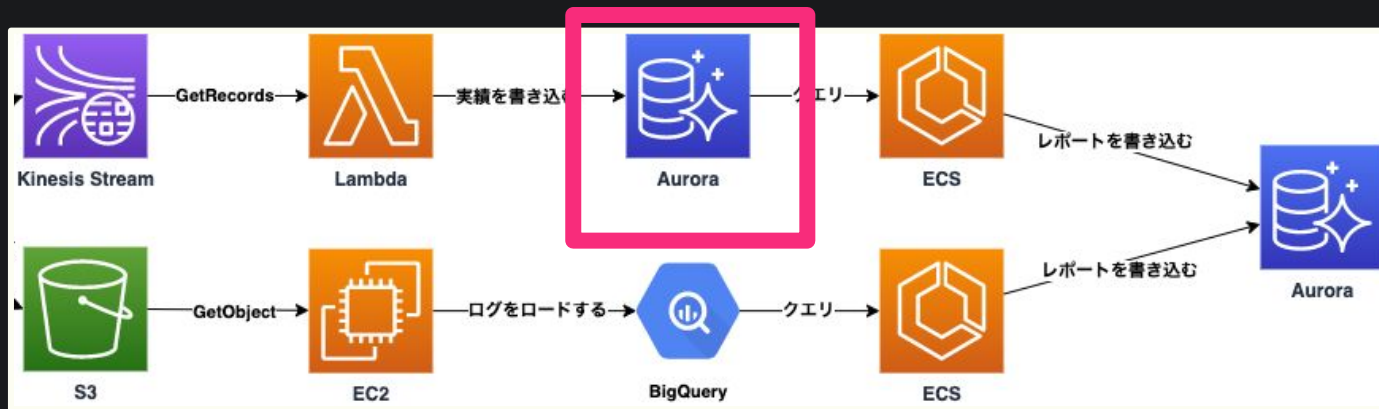
数字がズレると何が起きるか

混乱しか生まれない。

「なんでズレるんですか？」それは、私も知りたい。

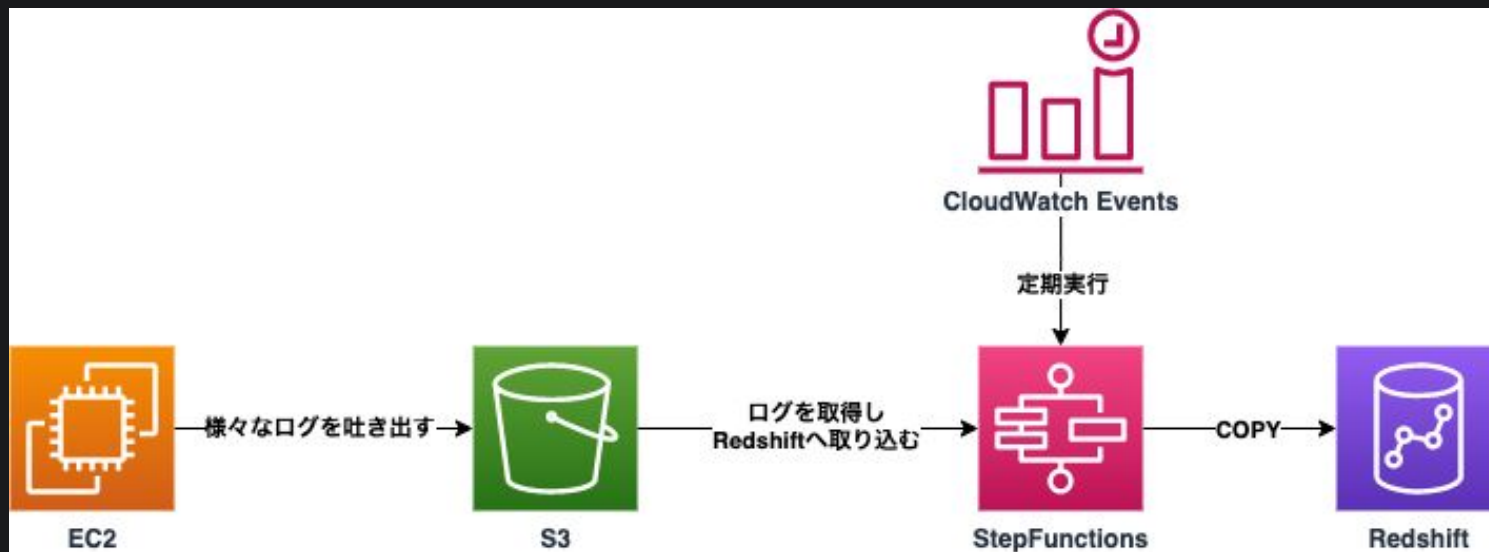
既存レポーティング基盤の見直し

ソースを一つにするにあたって、列指向データベースを採用した。
レポートにおいて、特定のクリック(行)に興味があることはなく、
「昨日、何回クリックが出たか？(列)」に興味があったため。



見直しの結果、次のようになる

新レポート基盤のアーキテクチャ



新レポーティング基盤のアーキテクチャ



S3にさえログがあれば復旧可能

ログさえ上げれば、レポートは復旧できる。

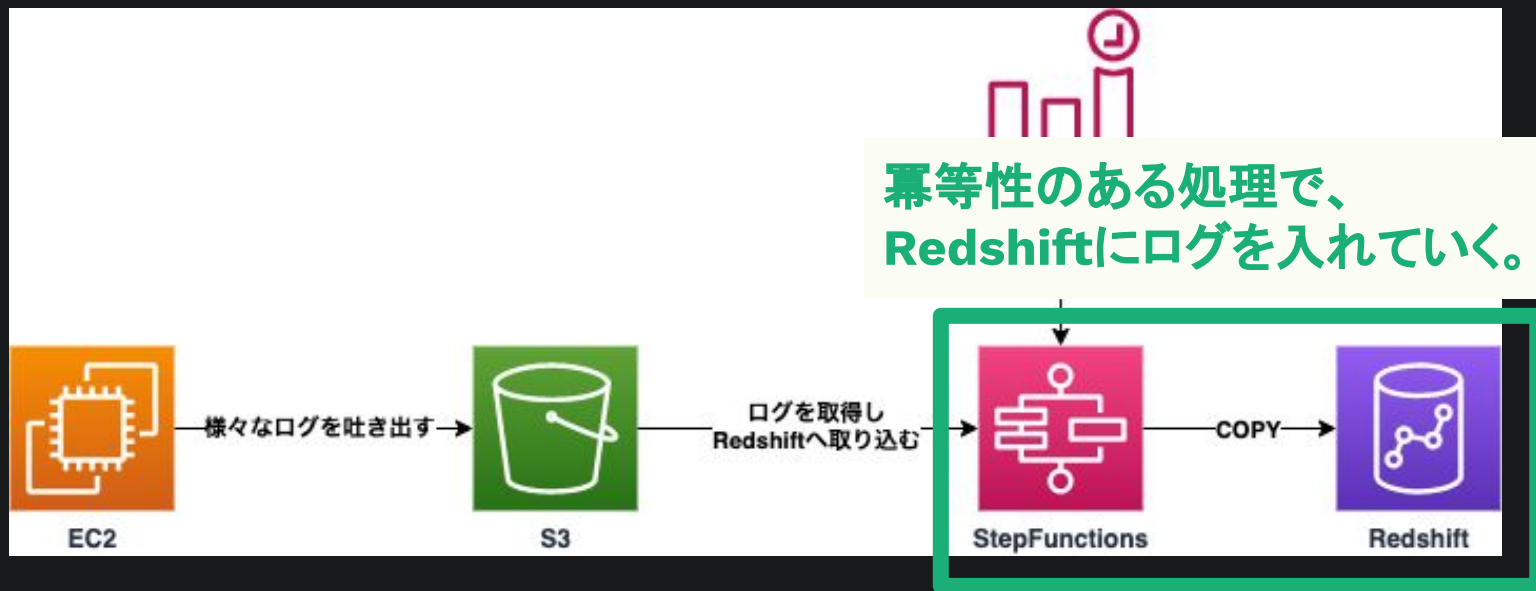
S3 は 99.999999999% (9 x 11) の耐久性がある安心感。

ログ = お金なので、とても重要。



Amazon S3
Simple Storage Service

新レポーティング基盤のアーキテクチャ



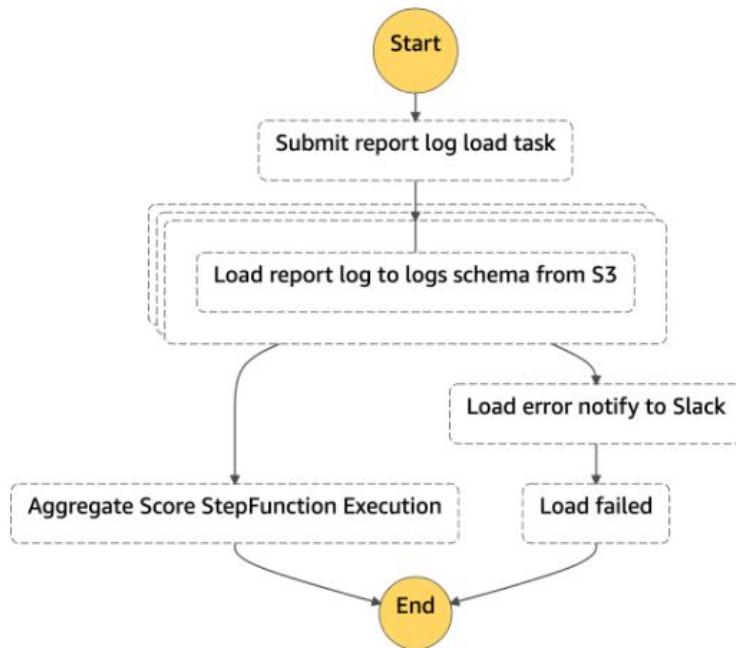
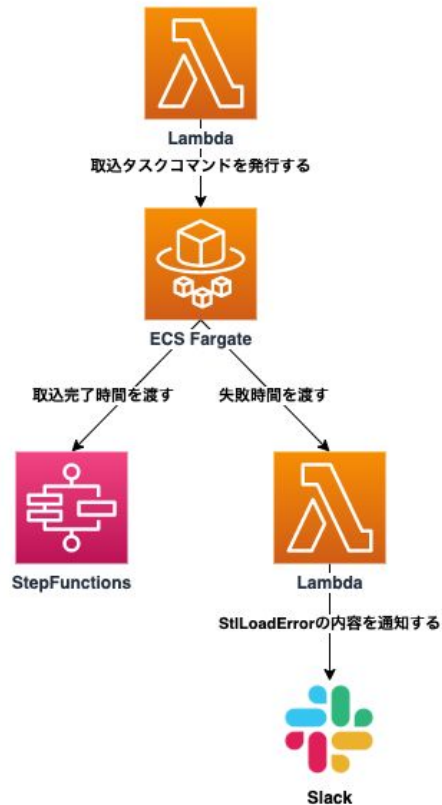
全てのデータ操作に冪等性を

処理全体に冪等性があれば、実装ミスがあっても、障害が起きて復旧する時も、様々な場面で難易度を下げられる。

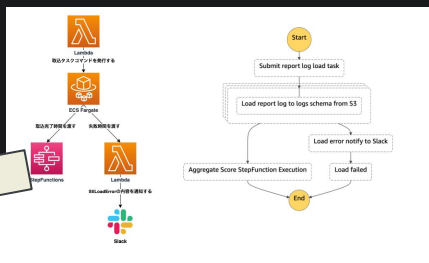
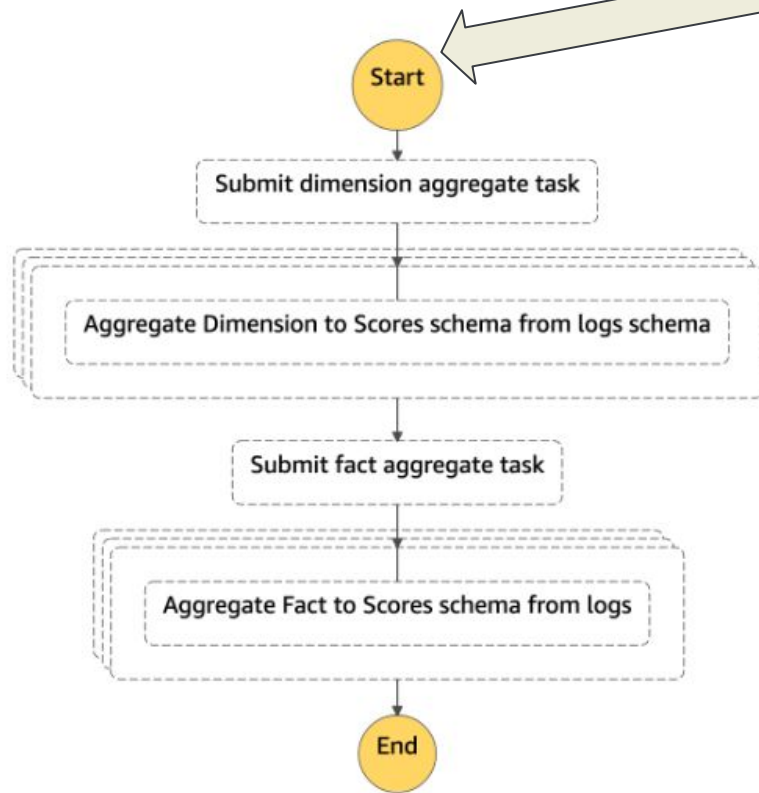
新しい基盤では、ログ取込から集計の**全てに冪等性を担保している**。

障害対応の時に繊細な再集計したくないよね？

ログ取込



集計



ログ取込

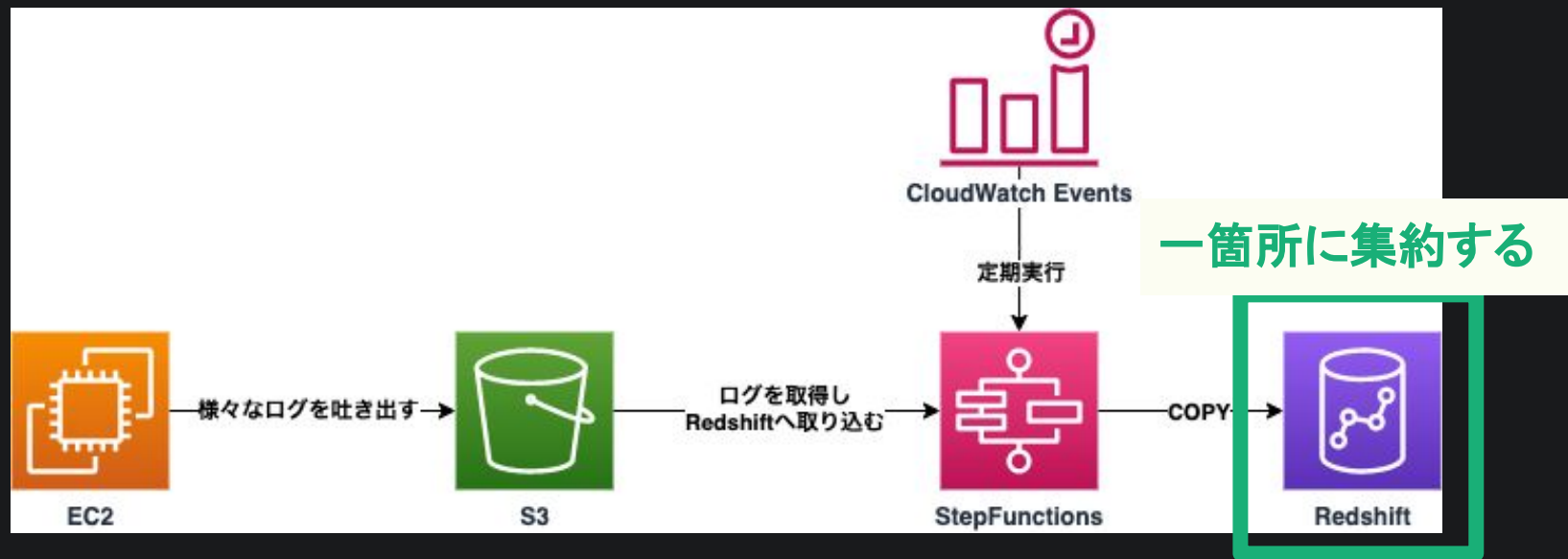
StepFunctions(SFN)の所感

- 機能面で困ったことはない。
 - リトライ、エラーハンドリング、並列処理、SFN->SFN。
- 安い。
 - 毎時で動かしてるけど、1日1ドルもかからない。
- ワークフローは、CDKで書けば書き味は悪くない。
 - ベタで書くのは地獄になると思う。
- サーバーレス。
 - フルマネージド最高。

SFNの設計で意識したこと

- ログ取込処理と集計処理は、それぞれで別SFNにした。
 - 処理ごとに、あとで変えやすくするため。
(ex: 取込をバッチからストリーミングへ)
 - 集計のみ再実行したいことがある。
- ECS FargateとRedshiftは落ちる前提。
 - 何らかの理由で落ちることがよくあるので、SFNで何度かリトライしてあげる。

新レポート基盤のアーキテクチャ



Redshiftの所感

- レポートと相性が良い。
 - 集計のパフォーマンスが高い。
 - キャッシュが強力(レポートは定形クエリ)
 - 今後AQUAで更に高速化されそう。
- COPYを使えば、簡単に大量のログを取り込めて便利。
 - S3にさえ上げてしまえば、ほとんど困らない。
- 自動テーブル最適化で、大体いい感じになる。
- フェデレーテッドクエリを使えば、AuroraのデータとJOINが出来る。
 - データを移動させなくていい。

バッチレイヤーと冪等性の組み合わせで、
安心感のあるデータ取り込みを実現。

早くレポートが見たい時は？

別でスピードレイヤーを組めば良いと考えている。

恐らくなるべく早く見たいレポートは、全ての指標ではなく、
一部の指標が見たいはずで、そこだけ集中的に対応すればよい。

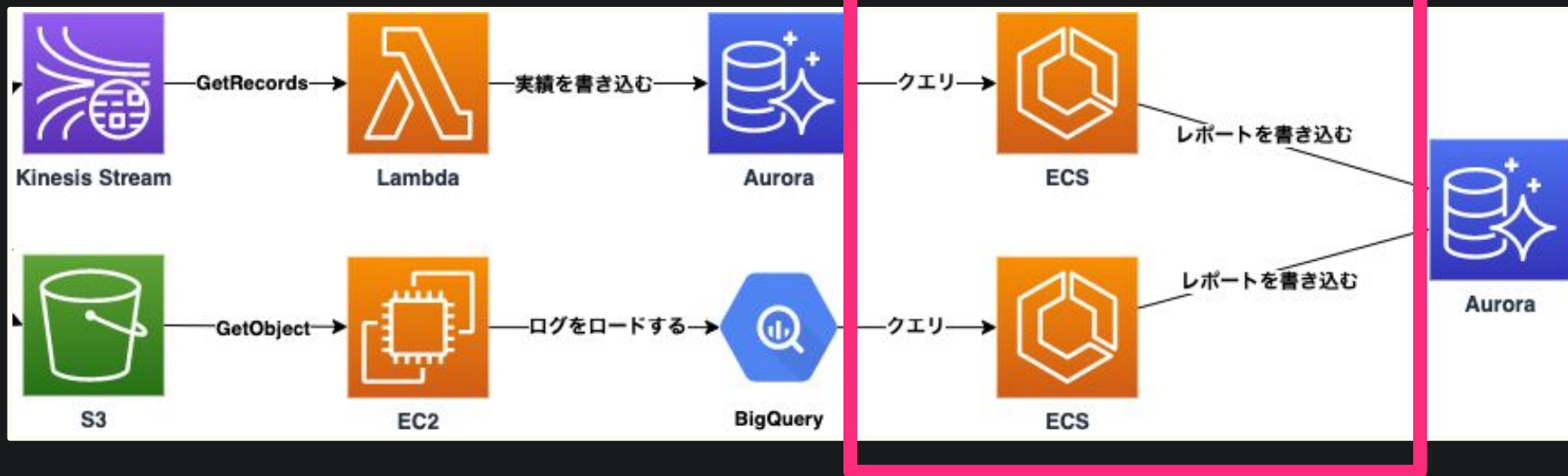
バッチレイヤーを頑張って10分ごとに反映するよりは、
特定のものに、スピードレイヤーで1分反映のが難易度は低いはず。
しかも、最終的にはバッチレイヤーのデータを正とするので、
捨てるデータになり、本当に速さ優先で組める。

アジェンダ

1. 自己紹介
2. レポーティング基盤について
3. 大量データとの戦い
4. 多様なデータとの戦い
5. まとめ

既存レポーティング基盤の課題

クエリがひたすらに辛い。



どんなクエリが走っているのか？

多様なデータとの戦い方

軸

- 日付
- 広告主ID
- キャンペーンID
- 代理店ID
- etc...

指標

- 広告表示回数
- クリック数
- コンバージョン数
- 広告単価
- etc...

多様なデータとの戦い方

軸

- 日付
- 広告主ID
- キャンペーンID
- 代理店ID
- etc...

指標

- 広告表示回数
- クリック数
- コンバージョン数
- 広告単価
- etc...

多様なデータとの戦い方

軸

- 日付

指標

- 広告表示回数

今月の広告の表示回数とクリック数を
広告主IDとキャンペーンIDごとに見たい。

- 代理店ID
- etc...

- 広告単価
- etc...

ポジションによって、見たい粒度が変わる。
そのため、色んな粒度で見られる。

ほしいレポートごとに
場当たりの的に対応してると、**本当に辛くなる。**

データがAuroraにあるから、
クエリ辛いの？

安心してください。何を使っても辛い。



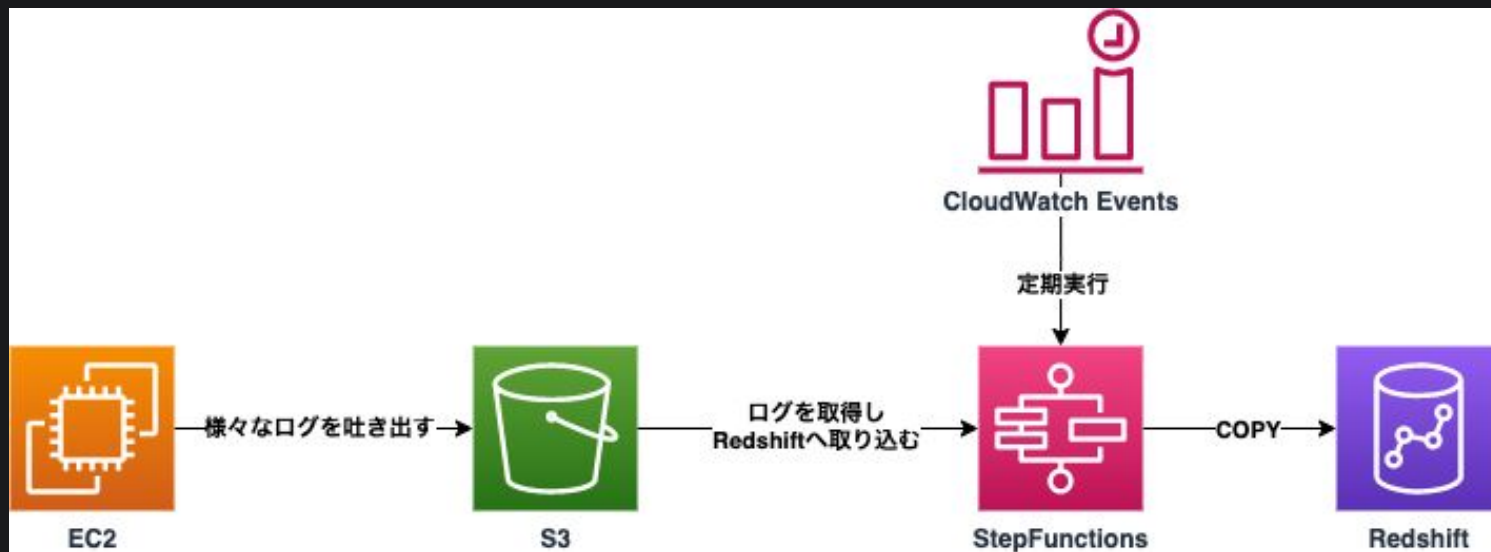
BigQuery



Amazon
Redshift



え？新アーキテクチャでも辛いの？



(再掲)
安心してください。何を使っても辛い。



BigQuery



Amazon
Redshift

 snowflake®

今日は、マシになる方法を紹介します。

なぜ、クエリが辛くなるのか。

色んなレポートに対応するから？

NO。核心はもっと深いところ。

ビジネスプロセスがモデリングされてないため、
ビジネスプロセスを取得するのに、一手間が入るから。

ビジネスプロセスとは

商品の発注を受けたり、誰かに送金などの組織が行う業務活動のこと。
DSPだと、広告表示された。広告がクリックされた。広告から商品が
購入されたなどがビジネスプロセスになる。

ビジネスプロセスがモデリングされてないあるある

テーブルにはtypeカラムがあり、これによって
どのビジネスプロセスか判定出来る。

- 1だったら、広告表示
- 2だったら、コンバージョン
- 3だったら、クリック
- 4だったら、etc...

ビジネスプロセスがモデリングされていないあるある

例えば、クリック数を取ろうとすると、typeを”3”で絞り込む必要がある。
つまり、**全てのクエリにWHERE句による絞り込みが必要になる。**

```
SELECT SUM(amount)  
FROM 実績 WHERE type = 3
```



※3はクリック

ビジネスプロセスがモデリングされてないあるある

一つのテーブルに色んなビジネスプロセスが入っていると・・・😂

- それぞれのカラムが、ビジネスプロセスごとに、使われ方が違うかもしれない。
- テーブルに変更を加えると、関係するビジネスプロセスに影響するかもしれない。
- 特定のtypeでしか使われないカラム。

こういった積み重ねが、
クエリを辛くする。

こういった辛さは、
モデリングを工夫すれば避けられる。

ディメンションモデリングを使えば🤪

こんな経験はありませんか？

- SELECTするのに、ドメイン知識が必要。
 - 「クリック？typeカラムを3にしたら取れるよ！」
- JOINの順番を工夫しないとだめ。
 - 間違えると、数字が爆増する。
- SQLによる計算がそこら中にハードコードされている。
 - 同じ指標がそこら中で計算されてる。(ロジックの分散)

上記の様なことが絡み合って、**クエリが辛くなる。**

こんな経験はありませんか？

- SELECTするのに、ドメイン知識が必要。
 - 「クリック？typeカラムを3にしたら取れるよ！」

少なくとも、こういうことは防げる。

○ 計算によって導かれるものは力があるけど ○

上記の様なことが絡み合って、クエリが辛くなる。

ディメンションモデリングとは

ビジネスプロセスがどのように測定されるかをモデル化することにより、分析可能にする。

<https://zenn.dev/pei0804/articles/dimensional-modeling>

VOYAGE GROUP Techlog Advent Calendar 2020 13日目

ディメンション・モデリングとは

ディメンション・モデリング

Wikipediaには以下のような説明がある。

Dimensional Modeling (DM) is a data structure technique optimized for data storage in a Data warehouse.

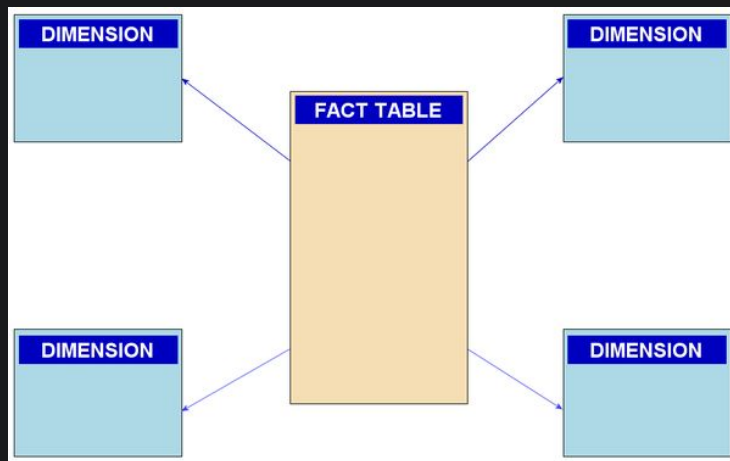
データウェアハウスにデータを格納するために、最適化されたデータ構造の手法。

背景

情報システムは2つの大きなカテゴリに分類される。1つはビジネスプロセスの実行支援する業務システム、もう1つはビジネスプロセスを分析支援する分析システム。それぞれ根本的に異なる目的があるため、異なる原則に基づき設計が進化してきた。

ディメンションモデリング in RDBMS

ディメンションモデリングをRDBMSで適用すると、
スタースキーマと呼ばれる設計になる。





スタースキーマ(基礎)

📅 2020.12.30に公開 🔄 2021.06.01に更新 ⌚ 7 min read



モデリング



データベース

データウェアハウス

スタースキーマ



Tech

[スタースキーマ wikipedia](#)

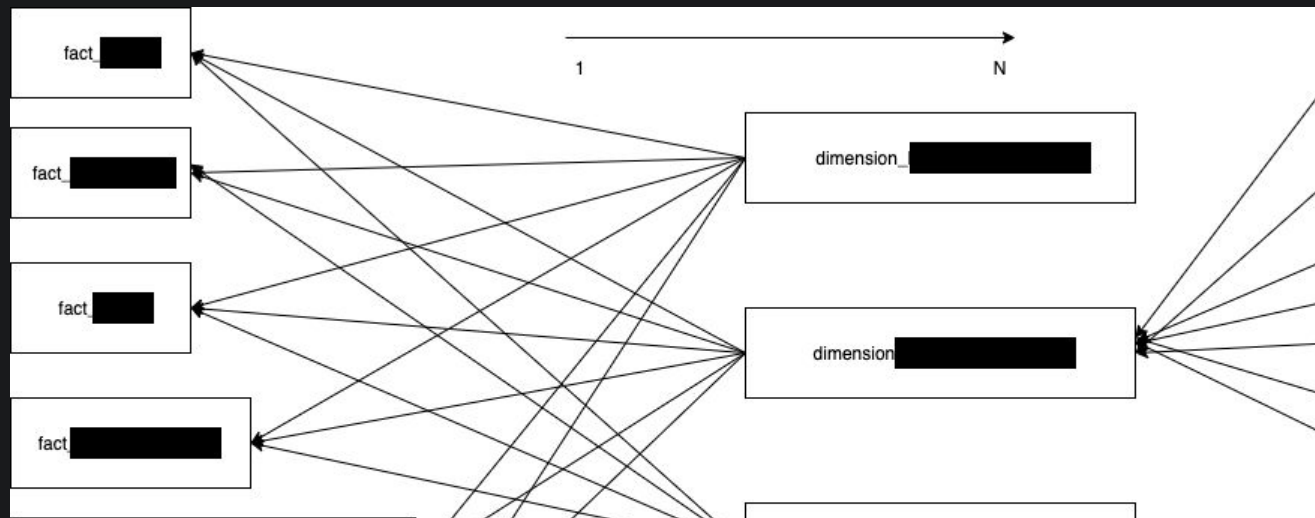
スタースキーマ または 星型スキーマ はデータウェアハウスに利用される最も単純なスキーマである。スタースキーマには唯1つもしくは少数のファクト表と複数のディメンション表が含まれる。スタースキーマはスノーフレイクスキーマの一種であるが、多くの用途で利用されている。

<https://zenn.dev/pei0804/articles/star-schema-design>

~~スタースキーマだと簡単にできる。~~

スタースキーマだと、**スケール可能になる。**

Real world Star Schema



闇雲なモデリングにはならないが、それなりに大変。

クリックのモデリング例

クリックのモデリングの例。

ログレベルのデータは、左のままでいいけど、

ELTなどで右のテーブルの状態に持っていけるのが望ましい。

元の実績テーブル
scored_at (発生日時)
type (種類)
advertiser_id (広告主ID)
creative_id (クリエイティブID)
様々な軸ID...
amount (量?)

モデリングしたクリックテーブル
click_key (クリックサロゲートキー)
scored_at (発生日時)
advertiser_id (広告主ID)
creative_id (クリエイティブID)
様々な軸ID...
click_count (クリック数)

良いモデリングはクエリもシンプルに

```
SELECT SUM(amount) FROM 実績 WHERE type = 3
```



```
SELECT SUM(click_count) FROM クリック
```

クエリでビジネスプロセスを表現するのではなく、
テーブルでビジネスプロセスを表現する。

弊社のスタースキーマの特徴

最終的に、全てを結合した大福帳テーブルを作成している。

基本的にレポートで必要とされる指標と軸は、そこに大体揃っているため、レポート画面からはそのテーブルへのクエリだけが行われる。

ちなみにRedshiftは、同じクエリには、キャッシュが効くので、よく使われるテーブルを意図的に作ると、かなりパフォーマンスが出る。

全てを網羅できているわけではない

ちなみに、大福帳テーブルで全てを網羅出来ているわけではない。
データ構造的に横に並べれないものは当然ある。そういったデータは
個別でレポートを出すか、UI側で頑張って横に並べるしかない。

銀の弾などない

スタースキーマの所感

- クエリとテーブルのパターン化が出来る。
 - 秩序を生み出せる。
- 業務システムのテーブル設計(正規化)とは考え方が全く違う。
- 様々な設計パターンを知る必要がある。
 - これは業務システムのテーブル設計でもそうかな。
- BIツールにそのまま読み込ませれるテーブルになる。
- 静的に構造を決定できるレポートとは相性が良い。

スタースキーマの学び方

日本語でまとまった情報少なかったので、
Zennにて少しずつ記事にしています。

<https://zenn.dev/pei0804>

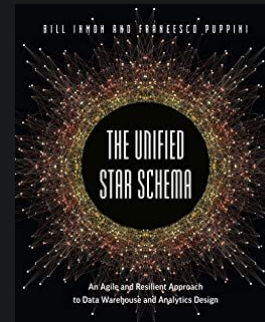
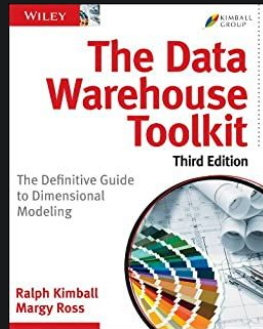
英語の記事はありますけど、文言くらいしかないので、
とりあえず、僕の記事読んだほうが分かりやすいです。

おすすめの読む順番

1. ディメンションモデリング
<https://zenn.dev/pei0804/articles/dimensional-modeling>
2. スタースキーマ(基礎)
<https://zenn.dev/pei0804/articles/star-schema-design>
3. 複数スタースキーマ
<https://zenn.dev/pei0804/articles/multiple-star-schema>
4. ファン・トラップ
<https://zenn.dev/pei0804/articles/datawarehouse-fan-trap>
5. コンフォームド・ディメンション
<https://zenn.dev/pei0804/articles/conformed-dimensions>
6. スロー・チェンジ・ディメンション
<https://zenn.dev/pei0804/articles/slowly-changing-dimensions>
7. スノーフレイクスキーマ
<https://zenn.dev/pei0804/articles/snowflake-schema>

気合があるならここらへんを読む

1. The Data Warehouse Toolkit: The Definitive Guide to Dimensional Modeling
2. Star Schema The Complete Reference
3. The Unified Star Schema:
An Agile and Resilient Approach to Data Warehouse and Analytics Design



楽しいレポートライフを👏

ところで、正しさの保証は出来る？

完全に正しさを保証することは難しい

せっかくモデリングしても、継続的に正しい状態になっているかを保証することは非常に難しい。

しかし、正しさの保証は重要です。なぜなら、その結果がお金に直結するからです。なので、クエリ変更でおかしくなることに気づきたい。ですが、愚直にテストとテストデータを用意するにしても、全てのパターンを網羅することは現実的ではないし、何故か落ちるテストになるのがオチ。

だが、絶望しないでほしい。異変に気付くコツはある。

レポートが壊れるパターン

レポートが壊れる時は、大体決まっている。

- WHERE句が間違えている。
- ファントラップが発生している。

そして、これらのミスが発生すると、生み出される結果は、
大体大きく数字がズれる。

これだけに気づけるようにすれば、ある程度のミスはカバー出来る。

ファントラップとは

1対多の関係のテーブルをJOINすると発生するトラップ。

JOINした結果、数字が増えた経験はありませんか？

それは、おそらくファントラップが起きています。

<https://zenn.dev/pei0804/articles/datawarehouse-fan-trap>



ソースと成果物を 比較する

クリックログが100クリックであるなら、
そこから作られた集計済みクリックテーブルの
結果も100クリックであるはず。
この結果を比較する。大きくズレてれば
何かが起きていると言える。
※弊社では、ファクトを集計してます。

集計前テーブルのクリック数。

```
select count(*) from logs.clicks
where
    ? <= request_time
    and request_time < ?
```

集計済みテーブルのクリック数。

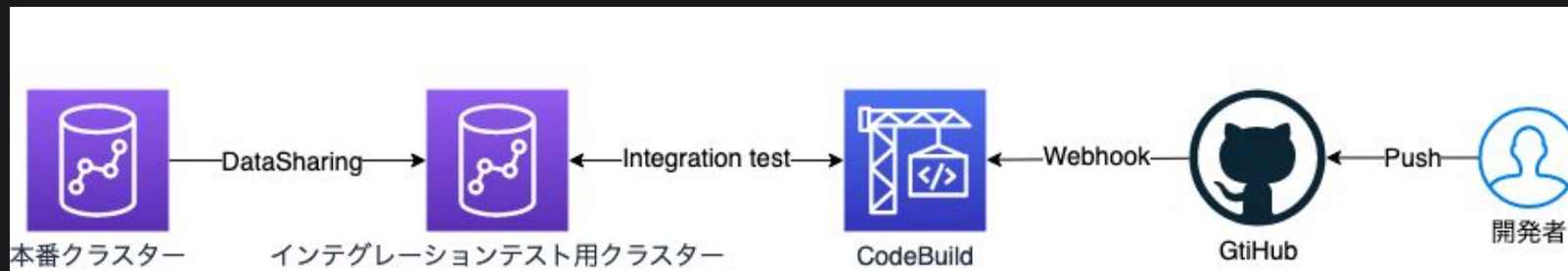
```
select sum(click_count) from scores.fact_click
where
    ? <= scored_at
    and scored_at < ?
```

テストデータは本物を使う

テストに使うデータは、**本物を使っている**。

ダミーデータではパターン漏れ発生するし、保守が現実的ではない。

本物のデータは、DataSharingで用意することで、容量を取らずに Read Onlyで本物のデータを別クラスターから読み込めるようになる。



アジェンダ

1. 自己紹介
2. レポーティング基盤について
3. 大量データとの戦い
4. 多様なデータとの戦い
5. まとめ

まとめ

- 適したアーキテクチャ。
 - 確定値がほしいレポーティングには、ラムダアーキテクチャのバッチレイヤーかつ冪等性で戦う。
 - なるべく早く見たいデータには、集中的にスピードレイヤーで対応する。
- 適したモデリング。
 - レポーティングの様な静的かつ構造化可能なものは、ディメンションモデリングが有効。